

Betriebssysteme (BS)

VL 8.4 – Koroutinen und Fäden – Multitasking

Volkmar Sieh / Daniel Lohmann

Lehrstuhl für Informatik 4
Verteilte Systeme und Betriebssysteme

Friedrich-Alexander-Universität
Erlangen Nürnberg

WS 21 – 13. Dezember 2021



https://www4.cs.fau.de/Lehre/WS21/V_BS

Agenda

Motivation

Grundbegriffe

Implementierung

Ausblick

Koroutinen als Hilfsmittel für das BS

Mehrfädrigkeit

Zusammenfassung

Referenzen



- Koroutinen sind (eigentlich) ein Sprachkonzept
 - Multitasking auf Sprachebene
 - wir haben es hier für C/C++ (bzw. ein ABI) „nachgerüstet“
 - Kontextwechsel erfordert keine Systemprivilegien!
 - ↪ muss also nicht zwingend im BS-Kern erfolgen
- Voraussetzung für echtes Multitasking: Kooperation
 - Anwendungen müssen als Koroutinen implementiert sein
 - Anwendungen müssen sich gegenseitig kennen
 - Anwendungen müssen sich gegenseitig aktivieren
 - ...



- Koroutinen sind (eigentlich) ein **Sprachkonzept**
 - Multitasking auf Sprachebene
 - wir haben es hier für C/C++ (bzw. ein ABI) „nachgerüstet“
 - Kontextwechsel erfordert keine Systemprivilegien!
 - ↪ muss also **nicht zwingend** im BS-Kern erfolgen
- Voraussetzung für echtes Multitasking: **Kooperation**
 - Anwendungen müssen als Koroutinen implementiert sein
 - Anwendungen müssen sich gegenseitig kennen
 - Anwendungen müssen sich gegenseitig aktivieren
 - ...

Problem

Für uneingeschränkten Mehrprogramm-Betrieb ist das **unrealistisch**.



Alternative: „Kooperationsfähigkeit“ als Aufgabe des Betriebssystems auffassen

Ansatz: Anwendungen „unbemerkt“ als eigenständige Fäden ausführen

- **BS** sorgt für die **Erzeugung** der Koroutinen-Kontrollflüsse
 - jede Anwendung wird als Routine aus einer **BS-Koroutine** aufgerufen
 - $\rightsquigarrow$ indirekt läuft jede Anwendung als Koroutine
- **BS** sorgt für die **Suspendierung** laufender Koroutinen-Kontrollflüsse
 - so dass Anwendungen nicht kooperieren müssen
 - erfordert einen **Verdrängungsmechanismus**
- **BS** sorgt für die **Auswahl** des nächsten Koroutinen-Kontrollflusses
 - so dass Anwendungen sich nicht gegenseitig kennen müssen
 - erfordert einen **Scheduler**



Alternative: „Kooperationsfähigkeit“ als Aufgabe des Betriebssystems auffassen

Ansatz: Anwendungen „unbemerkt“ als eigenständige Fäden ausführen

- **BS** sorgt für die **Erzeugung** der Koroutinen-Kontrollflüsse
 - jede Anwendung wird als Routine aus einer **BS-Koroutine** aufgerufen
 - $\rightsquigarrow$ indirekt läuft jede Anwendung als Koroutine
 - **BS** sorgt für die **Suspendierung** laufender Koroutinen-Kontrollflüsse
 - so dass Anwendungen nicht kooperieren müssen
 - erfordert einen **Verdrängungsmechanismus**
 - **BS** sorgt für die **Auswahl** des nächsten Koroutinen-Kontrollflusses
 - so dass Anwendungen sich nicht gegenseitig kennen müssen
 - erfordert einen **Scheduler**
- Mehr dazu in der nächsten Vorlesung!



Agenda

Motivation

Grundbegriffe

Implementierung

Ausblick

Zusammenfassung

Referenzen



- Ziel war die Ermöglichung von „Quasi-Parallelität“
 - Verschränkte Ausführung von Funktionen
 - Suspendierung und Reaktivierung von Funktions-Ausführungen
 - Begriff der Fortsetzung
- Routinen $\mapsto$ asymmetrisches Fortsetzungsmodell
 - Ausführung nach LIFO (und damit nicht „quasi-parallel“)
 - CPU und Übersetzer stellen Elementaroperationen bereit
- Koroutinen $\mapsto$ symmetrisches Fortsetzungsmodell
 - Ausführung in beliebiger Reihenfolge
 - erfordert eigenen Kontext: minimal PC, i. a. auch Register und Stapel
 - CPU und Übersetzer stellen i. a. keine Elementaroperationen bereit
- Fäden $\mapsto$ vom BS verwaltete Koroutinen

